**Instruction size:** 15 bits

* **Opcode:** 5 bit
* **Registers:** 3 bit
* **Memory addresses, Immediate values, Label names:** 4 bit

**Number of registers:** 8 registers in total since they are represented by 3 bits

**Memory size:** 4 bit addresses i.e. 24 = 16 bytes of memory

**Instructions:**

* **Init:** Initializes all register values to 0. Must be included at the start of every program.

|  |  |
| --- | --- |
| **Instruction format** | |
| Opcode | Zero filling |
| **Example** | |
| init |  |
| 00000 | 0000000000 |

* **Halt:** Indicates the end of the program. Program will finish execution either when halt is called or if the program counter reaches the end of the instructions.

|  |  |
| --- | --- |
| **Instruction format** | |
| Opcode | Zero filling |
| **Example** | |
| halt |  |
| 11111 | 0000000000 |

* **Input:** Takes user input and stores the value in the specified register.

|  |  |  |
| --- | --- | --- |
| **Instruction format** | | |
| Opcode | Register | Zero filling |
| **Example** | | |
| input | r0 |  |
| 00001 | 000 | 0000000 |

* **Output:** Prints the value of the specified register on the console.

|  |  |  |
| --- | --- | --- |
| **Instruction format** | | |
| Opcode | Register | Zero filling |
| **Example** | | |
| output | r0 |  |
| 00010 | 000 | 0000000 |

* **Add:** Takes values from two registers and stores their sum in another register.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Instruction format** | | | | |
| Opcode | Destination | Source1 | Source2 | Zero filling |
| **Example** | | | | |
| add | r0 | r1 | r2 |  |
| 00011 | 000 | 001 | 010 | 0 |

* **Sub:** Takes values from two registers and stores their difference in another register.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Instruction format** | | | | |
| Opcode | Destination | Source1 | Source2 | Zero filling |
| **Example** | | | | |
| sub | r0 | r1 | r2 |  |
| 00100 | 000 | 001 | 010 | 0 |

* **Mul:** Takes values from two registers and stores their product in another register.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Instruction format** | | | | |
| Opcode | Destination | Source1 | Source2 | Zero filling |
| **Example** | | | | |
| mul | r0 | r1 | r2 |  |
| 00101 | 000 | 001 | 010 | 0 |

* **Div:** Takes values from two registers and stores their quotient in another register.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Instruction format** | | | | |
| Opcode | Destination | Source1 | Source2 | Zero filling |
| **Example** | | | | |
| div | r0 | r1 | r2 |  |
| 00110 | 000 | 001 | 010 | 0 |

* **Jump:** Changes program counter to point at the start of a label.

|  |  |  |
| --- | --- | --- |
| **Instruction format** | | |
| Opcode | Label | Zero filling |
| **Example** | | |
| jump | label |  |
| 00111 | 0000 | 000000 |

* **Addi:** Stores the sum of a value from a register and an immediate value in the destination register.

|  |  |  |  |
| --- | --- | --- | --- |
| **Instruction format** | | | |
| Opcode | Destination | Source1 | Value |
| **Example** | | | |
| addi | r0 | r1 | 0 |
| 00011 | 000 | 001 | 0000 |

* **Im:** Loads an immediate value to a register.

|  |  |  |  |
| --- | --- | --- | --- |
| **Instruction format** | | | |
| Opcode | Register | Value | Zero filling |
| **Example** | | | |
| im | r0 | 0 |  |
| 01001 | 000 | 0000 | 000 |

* **Load:** Loads a value from an array at given offset in a register.

|  |  |  |  |
| --- | --- | --- | --- |
| **Instruction format** | | | |
| Opcode | Destination | Variable | Offset |
| **Example** | | | |
| load | r0 | var: | r1 |
| 01010 | 000 | 0000 | 001 |

* **Store:** Takes a value from a register and stores it in an array at an offset.

|  |  |  |  |
| --- | --- | --- | --- |
| **Instruction format** | | | |
| Opcode | Variable | Offset | Source |
| **Example** | | | |
| store | var: | r0 | r1 |
| 01011 | 0000 | 0000 | 001 |

* **Array:** Creates an array in memory of given size. The size is defined in a register.The array name can be any unique combination of 4 bits.

|  |  |  |  |
| --- | --- | --- | --- |
| **Instruction format** | | | |
| Opcode | Variable | Size | Zero filling |
| **Example** | | | |
| array | var | r0 |  |
| 01100 | 0000 | 000 | 000 |

* **>:** Executes the next instruction only if the value in register1 is greater than the value in register2. Else it skips the next instruction.

|  |  |  |  |
| --- | --- | --- | --- |
| **Instruction format** | | | |
| Opcode | Register1 | Register2 | Zero filling |
| **Example** | | | |
| > | r0 | r1 |  |
| 01101 | 000 | 001 | 0000 |

* **<:** Executes the next instruction only if the value in register1 is less than the value in register2. Else it skips the next instruction.

|  |  |  |  |
| --- | --- | --- | --- |
| **Instruction format** | | | |
| Opcode | Register1 | Register2 | Zero filling |
| **Example** | | | |
| < | r0 | r1 |  |
| 01110 | 000 | 001 | 0000 |

* **=:** Executes the next instruction only if the value in register1 is equal to the value in register2. Else it skips the next instruction.

|  |  |  |  |
| --- | --- | --- | --- |
| **Instruction format** | | | |
| Opcode | Register1 | Register2 | Zero filling |
| **Example** | | | |
| = | r0 | r1 |  |
| 01111 | 000 | 001 | 0000 |

* **!=:** Executes the next instruction only if the value in register1 is not equal the value in register2. Else it skips the next instruction.

|  |  |  |  |
| --- | --- | --- | --- |
| **Instruction format** | | | |
| Opcode | Register1 | Register2 | Zero filling |
| **Example** | | | |
| != | r0 | r1 |  |
| 10000 | 000 | 001 | 0000 |

* **Label:** Defines the start of a label. The label name can be any unique combination of 4 bits.

|  |  |  |
| --- | --- | --- |
| **Instruction format** | | |
| Opcode | Label name | Zero filling |
| **Example** | | |
| label | L: |  |
| 10001 | 0000 | 000000 |

**Registers:**

|  |  |
| --- | --- |
| r0 | 000 |
| r1 | 001 |
| r2 | 010 |
| r3 | 011 |
| r4 | 100 |
| r5 | 101 |
| r6 | 110 |
| r7 | 111 |

**Benchmark program:**

The program should have a .asm extension

**init**

**input** r0

**array** a, r0

**im** r1, 0

**label** loop:

**input** r2

**store** a:r1, r2

**addi** r1,r1,1

**>** r1, r0

**jump** exit\_loop

**jump** loop

**label** exit\_loop:

**im** r2, 0

**im** r3, 0

**label** sum:

**load** r1, a:r2

**add** r3, r3, r1

**addi** r2, r2, 1

**>** r2, r0

**jump** exit\_sum

**jump** sum

**label** exit\_sum:

**output** r3

**halt**

After assembling a .obj file should be created.

000000000000000

000010000000000

011000000000000

010010010000000

100010001000000

000010100000000

010110000001010

010000010010001

011010010000000

001110010000000

001110001000000

100010010000000

010010100000000

010010110000000

100010011000000

010100010000010

000110110110010

010000100100001

011010100000000

001110100000000

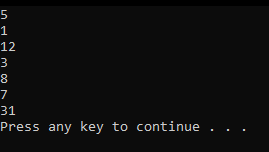
001110011000000

100010100000000

000100110000000

111110000000000

**Sample output:**

****